# Streamline-Computing
# Users Guide

Streamline Computing The Innovation Centre
Warwick Technology Park
Gallows Hill
Warwick CV34 6UW
http://www.streamline-computing.com
sales@streamline-computing.com
support@streamline-computing.com
Reference no: Feb 2010

Feb 2010

# Contents

**1 Introduction** **4**

**2 Logging in** **4**

**3 Modules: Re-setting the default environment** **5**
3.1 Preserving Modules environment across logins . . . . . 6

**4 Compilers: Gnu, Intel, PGI, Pathscale** **6**

**5 The SGE job scheduler** **7**
5.1 qsub: Submitting a simple job . . . . . 7
5.2 qstat: Querying the job queue . . . . . 8
5.3 qdel: Deleting a job . . . . . 9
5.4 Array jobs . . . . . 11
5.5 Submitting Dependent jobs . . . . . 12
5.6 Common SGE commands . . . . . 13

**6 Compiling and running OpenMP threaded applications** **14**
6.1 Compiling OpenMP code . . . . . 14
6.1.1 Gnu . . . . . 14
6.1.2 Intel . . . . . 15
6.1.3 Pgi . . . . . 15
6.1.4 Pathscale . . . . . 15
6.2 Running OpenMP code . . . . . 15

**7 Compiling and Running MPI codes** **17**
7.1 OpenMPI . . . . . 19
7.1.1 OpenMPI: selecting an interconnect . . . . . 19
7.1.2 compiling using OpenMPI . . . . . 20
7.1.3 submitting OpenMPI jobs . . . . . 21
7.2 SunHPC . . . . . 21
7.3 Mpich . . . . . 23
7.3.1 compiling Mpich codes . . . . . 23
7.3.2 submitting Mpich jobs . . . . . 23
7.4 Mpich2 . . . . . 23
7.4.1 compiling Mpich2 codes . . . . . 24
7.4.2 submitting Mpich2 jobs . . . . . 24
7.5 Mvapich 1 and 2 . . . . . 24

**8 HPC libraries: FFTW, Scalapack, Lapack, Blas, Atlas, MKL, ACML** **25**
8.1 FFTW . . . . . 25
8.2 Scalapack . . . . . 25

8.3 Lapack/Blas . . . 25
8.4 Atlas . . . 25
8.5 MKL . . . 25
8.6 ACML . . . 26
8.7 Goto Blas . . . 26
8.8 Linking code with Scalapack/Lapack/Blas . . . 26
8.8.1 Atlas . . . 27
8.8.2 MKL Version 11.1 . . . 27
8.8.3 ACML . . . 27
8.8.4 Goto Blas . . . 27

**9 Case study: AMBER8 benchmark 28**
9.1 Setting the environment . . . 28
9.2 Compiling the source code . . . 28
9.3 Running the code . . . 30

**10 Understanding SGE queues 32**

**11 Understanding SGE PE's 32**

**12 Further documentation 34**
12.1 Compilers . . . 34
12.1.1 Gnu . . . 34
12.1.2 Intel . . . 34
12.1.3 Pgi . . . 34
12.2 SGE . . . 35
12.3 OPenMP . . . 35
12.4 OpenMPI . . . 35
12.5 Mpich . . . 35
12.6 Mpich2 . . . 35
12.7 Netlib . . . 35
12.8 FFTW . . . 35

**13 FAQS 36**

**14 Trouble shooting 36**

# 1 Introduction

This is a brief guide intended to help users start running jobs in a short space of time. It covers compiling, linking and submitting jobs to a Streamline Cluster. We recognise that there will be exceptional cases where users have specialised requirements, and as such this guide cannot cover every scenario, however we hope that it is sufficent for a majority of users. This guide covers scalar (non-parallel), smp parallel (OpenMP), and MPI (Mpich, Mpich2,OpenMPI,SunHPC) jobs. It does not cover commercial codes which have their own embedded MPI.

Here is a brief précis of compiling and running an OpenMPI distributed memory job.

- User logs into Front end server.
- User compiles code : mpif90 -O3 -o mpitest mpitest.f
- User submits a 16 cpu job : ompisub 16 ./mpitest

# 2 Logging in

Secure Shell (ssh) is the standard way to login to the front-end server. Your local system administrator should have set you up an account. A suitable default shell environment has been set up for you when we installed the software, including paths, in order that you can run the jobs straight away. Please be cautious about copying environment files (.login, .cshrc, etc) from another machine as this may override the default settings and render the system unusable for you (if this happens ask your system administrator to restore the default settings). It is possible to easily modify your environment using Environment Modules discussed later. This is necessary if, for instance, you want to run several different versions of the same software.

Once logged in, a user should first check that their account on the cluster is properly set up and working. Valid users on a cluster should be able to login to any of the compute nodes using either rsh (remote shell) or ssh *without* being asked for a password or passphrase. On most clusters the default names for the compute nodes are comp00, comp01, comp02, etc (cat /etc/hosts.equiv file if in doubt). So, for example, a valid user should be able to do this:

```
~> rsh comp00 pwd
/home/sccomp
~> ssh comp00 pwd
/home/sccomp
```

If you cannot rsh or ssh to a compute node then either there is a problem with your account or there is a problem with the front end server and you

will not be able to run jobs on the compute nodes. Please seek help from your administrator in this case before proceeding.

## 3 Modules: Re-setting the default environment

You can reset the default environment by using the environment module package. To see what modules are available type 'module avail' eg

```
~> module avail
---------------------- /usr/share/Modules/modulefiles ------------------------
atlas             modules                   mvapich/pgi/1.1.0     pgi/9.0
cuda              mpich/2-1.0.6p1-GF90      mvapich/pgi/2-1.2p1   pgi/9.0-4
dot               mpich2                    mx                    sunhpc/8.2.1/gnu
gcc42             mvapich/gcc/1.1.0         null                  sunhpc/8.2.1/intel
intel/compiler111 mvapich/gcc/2-1.2p1       open-mx               sunhpc/8.2.1/pathscale
local_libs        mvapich/intel/1.1.0       openmpi/1.3.4-1/gnu   sunhpc/8.2.1/pgi
mkl/11.1          mvapich/intel/2-1.2p1     openmpi/1.3.4-1/intel sunhpc/8.2.1/sun
module-cvs        mvapich/pathscale/1.1.0   openmpi/1.3.4-1/path  use.own
module-info       mvapich/pathscale/2-1.2p1 openmpi/1.3.4-1/pgi
```

The default environment can be modified by loading, unloading and switching modules. Module commands are listed using the module help command:

```
~> module help
  Modules Release 3.1.6 (Copyright GNU GPL v2 1991):
  Available Commands and Usage:
        + add|load              modulefile [modulefile ...]
        + rm|unload             modulefile [modulefile ...]
        + switch|swap           modulefile1 modulefile2
        + display|show          modulefile [modulefile ...]
        + avail                 [modulefile [modulefile ...]]
        + use [-a|--append]     dir [dir ...]
        + unuse                 dir [dir ...]
        + update
        + purge
        + list
        + clear
        + help                  [modulefile [modulefile ...]]
        + whatis                [modulefile [modulefile ...]]
        + apropos|keyword       string
        + initadd               modulefile [modulefile ...]
        + initprepend           modulefile [modulefile ...]
        + initrm                modulefile [modulefile ...]
        + initswitch            modulefile1 modulefile2
        + initlist
```

```
        + initclear
```

Using the above example modules swap Intel 10.1 compiler to Intel 9.1 compiler:

```
~> module list
Currently Loaded Modulefiles:
  1) local_libs                 3) intel/compiler101_x86_64
  2) openmpi/1.2.6-1/intel
~> which ifort
/opt/intel/compiler101/x86_64/bin/ifort
~> module swap intel/compiler101_x86_64 \
                              intel/compiler91_x86_64
~> which ifort
/opt/intel/compiler91/x86_64/bin/ifort
```

### 3.1 Preserving Modules environment across logins

To save the current modules environment for use at next login use the the save_module_env utility:

```
acorn:~$ save_module_env
Saving the following module environemnts into your login startup:
Currently Loaded Modulefiles:
  1) cuda                        3) intel/compiler110_intel64   5) atlas
  2) openmpi/1.3.1-1/gcc         4) mkl/10.1.1.019/em64t
Proceed (y/n)[N] ?
y
acorn:~$
```

## 4 Compilers: Gnu, Intel, PGI, Pathscale

All Streamline clusters have the GNU, Intel, PGI and Pathscale compilers installed. This allows you to run binaries which have been compiled with Intel, PGI and Pathscale compilers on other systems. In order to compile locally using Intel, PGI or Pathscale you need to ensure that a valid license has been installed. If a compiler license was ordered with your system then Streamline-Computing will have already installed the license file and tested the appropriate compiler before shipping. The following table gives the names of the compilers for each compiler suite:

| COMPILER | C | C++ | F77 | F90 |
|---|---|---|---|---|
| Gnu | gcc | g++ | gfortran | gfortran |
| Intel | icc | icpc | ifort | ifort |
| Pgi | pgcc | pgCC | pgf77 | pgf90 |
| Pathscale | pathcc | pathCC | pathf90 | pathf90 |

Man pages are available for all these - eg man ifort.

# 5 The SGE job scheduler

The Sun Grid Engine (SGE) job manager is the default and recommended way to run applications on a Streamline cluster. SGE can manage usage of nodes, allows jobs to be queued when all nodes are in use and is highly configurable. On a multiuser system SGE avoids conflicts in resource usage and is vital in order to maintain a high job throughput. Streamline-Computing have configured SGE to be easy to use, both for scalar and parallel jobs, so that users can get started very quickly. This guide contains a short introduction to using SGE. More experienced users may find the SGE User guide SGE6-User.pdf located under /opt/streamline/DOC/SGE6 on the front end useful in more complex situations.

## 5.1 qsub: Submitting a simple job

In order to submit a job to the SGE job queues, a job script must be prepared. The job script may be written in any scripting language installed on the cluster. However the first line of the job script must indicate which scripting language is being employed. For example a job script written in the tcsh or csh language must start with:

```
#!/usr/bin/tcsh
```

a bash script must start with

```
#!/bin/bash
```

a perl script must begin with

```
#!/usr/bin/perl
```

and so on. Each job script consists of a series of syntactically correct unix commands or scripting lines. A job script does not need execute permission and can have any valid file name or name extension.
It is highly recommended to include on the second line of the script:

```
#$ -V -cwd
```

A line starting #$ is ignored by all scripting languages but is interpreted by SGE as flags sent to the SGE qsub command. In this case the flag -V instructs SGE to use the environment in force when the job was submitted (e.g PATH, LD_LIBRARY_PATH etc) when the job runs on one or more of the compute nodes. Without the -V flag, all the local settings will be lost when the job runs. This is especially important if you modified your environment using Environment Modules. The -cwd flag instructs SGE to

run the job script in the same directory that you were in when you submitted the job. Without the -cwd flag the job will start running in the users home directory, which in almost all cases will be incorrect.

Here is a simple job script called test.csh :

```
#!/usr/bin/tcsh
#$ -V -cwd
echo This script is running on node
hostname
echo The date is
date
sleep 20
```

To submit the job simply qsub it:

```
~/benchmarks> qsub test.csh
Your job 698 ("test.csh") has been submitted
```

### 5.2 qstat: Querying the job queue

To query a job use the qstat command:

```
~/benchmarks> qstat
job-ID  prior   name       user       state submit/start at  queue   slots ja-task
-----------------------------------------------------------------------------
698 0.00000 test.csh   sccomp       qw    07/12/2008 08:57:35   1
```

This shows that the job is queued and waiting and has been given a Job ID of 698. Later on it will will be running:

```
~/benchmarks> qstat
job-ID  prior   name       user      state submit/start at   queue   slots ja-task
-----------------------------------------------------------------------------
698 0.55500 test.csh   sccomp       r     07/12/2008 08:57:44 serial.q@comp00
```

This shows that the job was accepted by the serial.q queue and is actually running on node comp00. If the job has finished it will disappear from the qstat output. By default the standard output and error from a job are redirected to files which have the same name as the job script appended with a .o and .e respectively plus the Job ID number. This can be modified with the -o and -e flags to the qsub command. If you want the error and output to appear in the same file then use the -j y flags. In the above example 2 files are created by the job:

```
~/benchmarks> cat test.csh.o698
This script is running on node
comp00
```

```
The date is
Sat Jul 12 08:57:44 BST 2008
~/benchmarks> cat test.csh.e698
```

For further options on qsub see the qsub man page.

### 5.3 qdel: Deleting a job

If you want to remove a queued or running job from the job queue, use the qdel command followed by the Job ID number - eg

```
~/benchmarks> qsub test.csh
Your job 700 ("test.csh") has been submitted
~/benchmarks> qdel 700
sccomp has deleted job 700
```

As mentioned earlier, options to qsub can either be given after the qsub command or embedded in the job script using #$ .

Here's another example - compiling an application bench3 using the Intel compiler and submitting it to the serial queue, enforcing a maximum run time of 10 minutes.

```
~/benchmarks> module load intel/compiler101_x86_64
~/benchmarks> module load atlas
~/benchmarks> ifort -O3 -axT -o bench3 bench3.f \
                 -L/usr/local/lib64/atlas -lcblas -lf77blas -latlas
bench3.f(42): (col. 8) remark: BLOCK WAS VECTORIZED.
bench3.f(134): (col. 8) remark: LOOP WAS VECTORIZED.
bench3.f(167): (col. 15) remark: LOOP WAS VECTORIZED.
bench3.f(191): (col. 10) remark: LOOP WAS VECTORIZED.
bench3.f(203): (col. 8) remark: LOOP WAS VECTORIZED.
bench3.f(212): (col. 20) remark: zinitvecs_ has been targeted for automatic cpu dispatch.
```

This is a simple job script bench3.sh :

```
#!/bin/bash
#$ -V -cwd
echo "Running on $(hostname)"
echo "Cpu info follows"
cat /proc/cpuinfo  | grep 'model name' | head -1
echo "Start time" `date`
./bench3
echo "End  time" `date`
```

Next, submit the job. In this example we made a request that the job should be killed if it lasts for more than 10 minutes by adding -l h_rt=00:10:00 to the qsub option (h_rt is hard real time) - see man qsub for the qsub -l option, and man 5 complex for a description of SGE resource attributes.

```
~/benchmarks> qsub bench3.sh -l h_rt=00:10:00 bench3.sh
Your job 205 ("bench3.sh -l h_rt=00:10:00 bench3.sh") has been submitted
~/benchmarks> qstat
job-ID  prior   name       user      state submit/start at     queue       slots j
-------------------------------------------------------------------------------
    205 0.55500 bench3.sh  sccomp       r     07/12/2008 10:35:05 serial.q@comp00
```

This is the job output file bench3.sh.o205 after the job has run:

```
Running on comp00
Cpu info follows
model name      : Intel(R) Xeon(R) CPU           E5420  @ 2.50GHz
Start time Sat Jul 12 10:35:05 BST 2008
  Working out sensible value of nflops
  for this cpu
  Bench with    2621.44000000000       Mflops
  Min time per test =  0.5400000
  Starting benchmark #1
==================================
 RAW CPU RATE =   9709.04 Mflops
==================================
  Starting benchmark Blas 3
======================================
      Blas 3 dgemm (Matrix * matrix)
======================================
 Matrix dim (nXn,n=)   Mflop rate
        8              1424.70
       16              2016.49
       32              2166.48
       64              5957.82
      128              6721.64
      256              7073.64
      512              7549.75
     1024              7669.58
     2048              7669.58
     4096              7631.26
End  time Sat Jul 12 10:35:33 BST 2008
```

### 5.4 Array jobs

Another powerful feature of SGE is the ability to submit "array" jobs. This allows a user to submit a range of jobs with a single qsub. For example:

```
  qsub  -t 1-100:2  myjob.sh
```

This will submit 50 tasks (1,3,5,7,...,99). The job script knows which of the tasks it is via the $SGE_TASK_ID variable.

For example a job script might look like:

```
#!/bin/sh
#$ -V -cwd
TASK=$SGE_TASK_ID
# Run my code for input case $TASK and output it to an
# appropriate output file.
cd /users/nrcb/data
DATE=‘date‘
echo "This is the standard output for task $TASK on $DATE"
/users/nrcb/bin/mycode.exe input.$TASK output.$TASK
```

This would enable a user to run the code mycode.exe taking it's input from a series of input files input.1, input.3,...,input.99 and sending the output of the run to output files output.1,...,output.99.

### 5.5 Submitting Dependent jobs

In some cases a user requires to run a series of different codes on some initial data. The results of the previous job are used as inputs to the next job in the series. For example in many engineering calculations a pre-processor must be run on a user defined input file to generate a set of grid and flow files. These files are then used as the inputs to the main calculation. The main calculation cannot start until the pre-processor has run. SGE can deal with this scenario easily using dependent jobs. This is explained by the following example. Suppose we have a job script A.sh which must run before script B.sh. First we submit script A.sh and tag it with a name (jobA in this example) as follows:

```
[ benchmarks]$ qsub -N jobA A.sh
Your job 22 ("jobA") has been submitted
```

We then submit the B.sh script and tell it not to run until jobA has completed.

```
[ benchmarks]$ qsub -hold_jid jobA B.sh
Your job 23 ("B.sh") has been submitted
[ benchmarks]$ qstat
------------------------------------------------------------------------------
 22 0.55500 jobA       sccomp       r     05/16/2009 16:07:30 serial.q@comp00
 23 0.00000 B.sh       sccomp       hqw   05/16/2009 16:07:35
```

In the above jobA (the script A.sh) is running, and B.sh is held. The ouput of jobA is jobA.o22. When jobA has finished, the hold on job 23 is automatically released.

```
[ benchmarks]$ qstat
------------------------------------------------------------------------------
 23 0.00000 B.sh       sccomp       qw    05/16/2009 16:07:35
```

Finally B.sh runs.

```
[ benchmarks]$ qstat
------------------------------------------------------------------------------
 23 0.55500 B.sh       sccomp       r     05/16/2009 16:07:54 serial.q@comp00
```

Job dependency can be used with array jobs. For example (taken from the GridEngine user guide):

```
$ qsub -t 1-3 A
$ qsub -hold_jid A -t 1-3 B
```

All the sub-tasks in job B will wait for all sub-tasks 1,2 and 3 in A to finish before starting the tasks in job B. An additional facility with dependent array jobs is the ability to order the dependencies of the the individual array tasks. For example:

```
$ qsub -t 1-3 A
$ qsub -hold_jid_ad A -t 1-3 B
```

Sub-task B.1 will only start when A.1 completes. B.2 will only start once A.2 completes, etc.

### 5.6 Common SGE commands

Here are some of the more commonly used user commands for examining and manipulating SGE jobs:

| Command | action |
|---|---|
| qstat | queries queues for status of all jobs |
| qstat -f | qstat with verbose (full) output |
| qstat -u *username* | checks all queues for status of *username*'s jobs |
| qstat -g c | checks status of all queues |
| qstat -g c -q *queue* | checks status of queue *queue* |
| qstat -j JOB_ID | queries job JOB_ID |
| qdel JOB_ID | delete job JOB_ID |
| qdel a b c d e .. | delete jobs JOB_ID=a,b,c,d,e.. |
| qdel -u *username* | delete all *username*'s jobs |
| qhold JOB_ID | hold queued job JOB_ID |
| qhold -u *username* | hold all queued jobs belonging to *username* |
| qrls JOB_ID | release the hold on queued job JOB_ID |
| qrls -u *username* | release holds on all queued jobs belonging to *username* |

For more advanced options see man pages for qsub, qdel, qhold, qrls, qalter.

# 6 Compiling and running OpenMP threaded applications

Code with embedded OpenMP directives may be compiled and run on a single compute node with up to a maximum of NCORES threads via the smp parallel environment, where NCORES is the number of cpu cores per node.

## 6.1 Compiling OpenMP code

OpenMP code may be compiled with Intel, Pgi and Pathscale compilers and with gcc/gfortran version >= 4.2 . Here is a simple example from the tutorial at

```
http://openmp.org/wp/
```

```
       program hello
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c Reference: https://computing.llnl.gov/tutorials/openMP/ c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
       integer nthreads, tid, omp_get_num_threads,
     +   omp_get_thread_num
c fork a team of threads giving them their own copies of variables
!$omp parallel private(tid)
c obtain and print thread id
      tid = omp_get_thread_num()
      print *, 'hello world from thread = ', tid
c only master thread does this
      if (tid .eq. 0) then
        nthreads = omp_get_num_threads()
        print *, 'number of threads = ', nthreads
      end if
c all threads join master thread and disband
!$omp end parallel
       end
```

Here are basic compile options for Gnu, Intel, Pgi and Pathscale compilers for Fortran code (C code same - but substitute corresponding C compiler in each case).

### 6.1.1 Gnu

```
gfortran -fopenmp -o hello omphello.f
```

### 6.1.2 Intel

```
ifort -openmp -o hello omphello.f
```

### 6.1.3 Pgi

```
pgf90 -mp -o hello omphello.f
```

### 6.1.4 Pathscale

```
pathf90 -mp -o hello omphello.f
```

## 6.2 Running OpenMP code

To run an OpenMP code, create a job script and submit it to the smp parallel environment. Using the hello example above here is a script called run.csh

```
#!/usr/bin/tcsh
#$ -V -cwd -pe smp 1
setenv OMP_NUM_THREADS 4
echo Running on ; hostname
./hello
```

The same job in bash/sh shell syntax is:

```
#!/bin/bash
#$ -V -cwd -pe smp 1
export OMP_NUM_THREADS=4
echo Running on ; hostname
./hello
```

To submit the script:

```
> qsub run.csh
Your job 208 ("run.csh") has been submitted
```

The output file run.csh.o208 after the job has completed :

```
Running on
comp07
 hello world from thread =            0
 number of threads =            4
 hello world from thread =            1
 hello world from thread =            3
 hello world from thread =            2
```

There are two points to note. Firstly the second line of the run.csh script contains -pe smp 1 . This ensures that the script is submitted to the smp parallel environment. Secondly the environment variable OMP_NUM_THREADS should be set to the number of required threads. It is recommended the value does not exceed the number of cores/cpus per node. If the OMP_NUM_THREADS variable is not set then the default value depends on which compiler was used according to the following table:

| COMPILER | default OMP_NUM_THREADS |
|---|---|
| Gnu | NCORES |
| Intel | NCORES |
| Pgi | 1 |
| Pathscale | NCORES |

Here is another example: a benchmark on 1,2,4,8 processors of the code jacobi_omp. Firstly compile the code using the Intel compiler:

```
ifort -O3 -axT -openmp -o jacobi_omp jacobi_omp.f
jacobi_omp.f(25): (col. 7) remark: OpenMP DEFINED REGION WAS PARALLELIZED.
jacobi_omp.f(21): (col. 14) remark: LOOP WAS VECTORIZED.
jacobi_omp.f(21): (col. 14) remark: LOOP WAS VECTORIZED.
jacobi_omp.f(35): (col. 17) remark: LOOP WAS VECTORIZED.
jacobi_omp.f(140): (col. 7) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
jacobi_omp.f(139): (col. 7) remark: OpenMP DEFINED REGION WAS PARALLELIZED.
jacobi_omp.f(142): (col. 11) remark: LOOP WAS VECTORIZED.
jacobi_omp.f(92): (col. 11) remark: LOOP WAS VECTORIZED.
jacobi_omp.f(99): (col. 11) remark: LOOP WAS VECTORIZED.
jacobi_omp.f(114): (col. 11) remark: LOOP WAS VECTORIZED.
```

Here is the job script, jacobi_run.sh:

```
#!/bin/bash
#$ -V -cwd -pe smp 1 -j y
THREADS="1 2 4 8"
echo " Running on node ‘hostname‘"
jstart=‘date +%s‘
OUTFILE=jacobi.out
rm $OUTFILE
echo " CPUS  Parallel Speed up"
for t in $THREADS; do
    export OMP_NUM_THREADS=$t
    start=‘date +%s‘
    ./jacobi_omp >> $OUTFILE
    end=‘date +%s‘
    time=$((end-start))
    if [ "$OMP_NUM_THREADS" -eq 1 ]; then
```

```
       time_for_1cpu=$time
    fi
    speedup=‘echo $time_for_1cpu/$time | bc -l -q‘
    echo "   $OMP_NUM_THREADS $speedup"
done
echo " Flop rates:"
grep MFlop $OUTFILE
jend=‘date +%s‘
echo "Total elapsed time = $((jend-jstart)) seconds"
```

Submit the job:

```
qsub jacobi_run.sh
Your job 241 ("jacobi_run.sh") has been submitted
```

The output file (and error file) jacobi_run.sh.o241:

```
Running on node comp02
 CPUS  Parallel Speed up
   1 1.00000000000000000000
   2 1.91071428571428571428
   4 3.68965517241379310344
   8 6.68750000000000000000
 Flop rates:
 cpus=  1 MFlop rate  = 7.76D+02
 cpus=  2 MFlop rate  = 1.52D+03
 cpus=  4 MFlop rate  = 2.86D+03
 cpus=  8 MFlop rate  = 5.40D+03
Total elapsed time = 208 seconds
```

Other threaded codes, for example the Chemistry code Gaussian, and the Engineering codes Abaqus and LS-Dyna

```
http://www.gaussian.com/
http://www.simulia.com/
http://www.lstc.com/
```

are all capable of running through the smp parallel environment. Please see the documentation which comes with such codes.

# 7 Compiling and Running MPI codes

A number of different versions of MPI may be available on your cluster. Depending on your hardware, not all of these MPI's may be operational. However it may still be possible to compile codes up using a non-operational MPI (eg for running on a different cluster). The following subsections describe

the main types of MPI to be found on Streamline clusters. Most versions of MPI provide a wrapper script for compiling codes. These wrapper scripts invoke an underlying compiler and, when linking, ensure the correct MPI libraries are loaded. For example compiling an f90 code with mpif90 is recommended over using ifort and linking the correct mpi libraries by hand.

In order to obtain maximum benefit from your cluster it is important that parallel MPI jobs be scheduled through Sun Grid Engine. Writing job scripts for parallel jobs can be tedious and error prone. For this reason Streamline Computing have developed a set of meta-scripts. These allow users to submit MPI parallel jobs very easily. Although these meta-scripts don't cover all eventualities, they are useful for 99% of jobs and on many systems are the only method employed for submitting jobs. Because of differences in the way various MPI's spawn parallel jobs a different meta-script is needed for each type of MPI. The general invocation of a meta-script is :

```
MSCRIPT NCPUS EXEC ["ARGS"]
```

where MSCRIPT is the name of the meta-script (described in following subsections) , NCPUS is the number of processes, EXEC is the full path of the binary executable and ARGS (in double quotes) are any arguments required by the executable. The number of processes may be input as a plain number - eg 16 or as nodes x cores - eg 8x2 . If you use a plain number the meta-script will generate a job script using the maximum number of cores per node. If you use the nodes x cores format the cores should not exceed the maximum number of cores in each compute node. In addition all meta-scripts accept the SGE_RESOURCES and QSUB_OPTIONS environment variables. The value of SGE_RESOURCES is added to the options to SGE's qsub -l option when the job is submitted. For example

```
export SGE_RESOURCES="bigmem"
export QSUB_OPTIONS="-a 02011200"
```

would add the options -a 02011200 -l bigmem to the qsub embedded in the meta-script.
Some applications require the input data to be redirected from a file using the unix < redirect. To do this with a meta-script, just pretend it is another argument. E,g

```
MSCRIPT NCPUS EXEC "ARGS < input "
```

The following meta-scripts and the versions of MPI they are used with, are listed below:

- mpisub : SCore MPI
- mpichsub : Myrinet mpich-mx, mpich, Infinipath mpi

- mpich2sub: mpich2
- ompisub : OpenMPI

Other advantages of using the Streamline meta-scripts are that they provide additional job information in the job output file: job execution time, a list of nodes the job ran on, and list of run time arguments used.

## 7.1 OpenMPI

Streamline-Computing clusters now come with OpenMPI compiled up for multiple compilers and multiple interconnects. In order to use OpenMPI you must therefore set up your environment to use the correct compiler and interconect. Please note that most clusters come equipped with at most 2 types of interconnect.

### 7.1.1 OpenMPI: selecting an interconnect

You can use the MCA-PARAMS setup script. Running the script gives useage:

```
cd /opt/streamline/MCA-PARAMS
./setup

 Installs mca-params.conf file for OpenMPI into ~/.openmpi

 Usage: setup [fabric]

 Where fabric is one of:

 eth0 eth1 eth2 eth3 mx psm openib omx.eth0 omx.eth1 omx.eth2 omx.eth3

 Example: ./setup eth0

====================       ====================
Interconnect               Params file
====================       ====================
TCP sockets on eth0        mca-params.conf.eth0
TCP sockets on eth1        mca-params.conf.eth1
TCP sockets on eth2        mca-params.conf.eth2
TCP sockets on eth3        mca-params.conf.eth3
Myrinet MX                 mca-params.conf.mx
Open-MX ethernet on eth0   mca-params.conf.omx.eth0
Open-MX ethernet on eth1   mca-params.conf.omx.eth1
Open-MX ethernet on eth2   mca-params.conf.omx.eth2
Open-MX ethernet on eth3   mca-params.conf.omx.eth3
```

```
Infiniband (eg Mellanox)  mca-params.conf.openib
Qlogic Infinipath         mca-params.conf.psm
```

For example to set up to use Infiniband :

```
./setup openib
 /home/nick/.openmpi/mca-params.conf exists.
 Overwrite (y/n) [N] ?
y
 Setting OpenMPI mca-params.conf.openib as default.
```

To set up to use Pathscale Infinipath:

```
./setup psm
 /home/nick/.openmpi/mca-params.conf exists.
 Overwrite (y/n) [N] ?
```

You can therefore change the interconnect at any stage. The setup program creates the file mca-params.conf in your .openmpi directory - hence a setup is permanent across logins. In order to compile codes using OpenMPI, first check that OpenMPI is in the path or change your environment. E.g : To use Gnu compilers:

```
~> module load openmpi/1.2.6-1/gcc
```

To use PGI compilers:

```
~> module load openmpi/1.2.6-1/pgi
```

To use Intel compilers:

```
~> module load openmpi/1.2.6-1/intel
```

To use Pathscale compilers:

```
~> module load openmpi/1.2.6-1/path
```

In addition to changing your environment to use the appropriate OpenMPI you may also need to make sure the compiler is also in your path.

### 7.1.2 compiling using OpenMPI

Here is an example using the Intel compiler.

```
~/benchmarks> module load openmpi/1.2.6-1/intel
~/benchmarks> module load intel/compiler101_x86_64
~/benchmarks> which mpif90
/opt/openmpi-1.2.6-1/intel/bin/mpif90
~/benchmarks> which ifort
/opt/intel/compiler101/x86_64/bin/ifort
~/benchmarks> mpif90 -O3 -axT -o mpitest mpitest.f
mpitest.f(24): (col. 12) remark: LOOP WAS VECTORIZED.
mpitest.f(96): (col. 9) remark: LOOP WAS VECTORIZED.
mpitest.f(52): (col. 9) remark: LOOP WAS VECTORIZED.
```

### 7.1.3 submitting OpenMPI jobs

It is recommended to use the ompisub meta-script as the following example demonstrates.

```
~/benchmarks> ompisub 16 ./mpitest

Generating SGE job file for a 16 cpu mpich job with SMP=8 from executable /users/sccomp/ben
QSUB mpirun -np 16 /users/sccomp/benchmarks/./mpitest

Done.

Submitting SGE job as follows:

qsub -pe openmpi 2 /users/sccomp/benchmarks/mpitest.sh

Sending standard output to file: /users/sccomp/benchmarks/mpitest.sh.o198
Sending standard error  to file: /users/sccomp/benchmarks/mpitest.sh.e198

Use the qstat command to query the job queue. e.g
qstat
job-ID  prior   name       user         state submit/start at     queue
-------------------------------------------------------------------------
    198 0.00000 mpitest.sh sccomp       qw    07/10/2008 19:10:34

Job submission complete.
```

The meta-script ompisub also adds the value of environment variable MPIRUN_ARGS to OpenMPI's mpirun. This can be used to change the behaviour of OpenMPI's mpirun as described in the OpenMPI documentation and FAQ's . See for instance http://www.open-mpi.org/faq/?category=running . For example to force OpenMPI to run over tcp on device eth0 (eg to test the difference between same code run over infiniband and over gigabit ethernet).

```
export MPIRUN_ARGS="--mca btl_tcp_if_include eth0 --mca btl tcp,self"
ompisub 16 ./mpitest
```

## 7.2 SunHPC

SunHPC Cluster Tools are Sun Microsystem's own MPI based on OpenMPI and are currently freely available for download. The compile and run instructions for OpenMPI carry through to SunHPC. In addition to the Intel, Gnu, PGI and Pathscale compiler support, Sun HPC also supports Sun's own Forte Compiler Suite. Another advantage of SunHPC over OpenMPI is that it simutaneously supports 32 and 64 bit compilation and runtime

with the same package set. SunHPC does not support Infinipath PSM at the time of writing. It does support Myricom and Open MX, Infiniband, and tcp.

In order to compile and run a SunHPC MPI job you first need to make sure you have selected the correct OpenMPI interconnect via the mca_params (see OpenMPI: selecting an interconnect). Next you must set up your environment for the required compiler support. E.g using the 8.2.1 version of SunHPC:
To use Gnu compilers:

```
~> module load sunhpc/8.2.1/gnu
```

To use PGI compilers:

```
~> module load sunhpc/8.2.1/pgi
```

To use Intel compilers:

```
~> module load sunhpc/8.2.1/intel
```

To use Pathscale compilers:

```
~> module load sunhpc/8.2.1/pathscale
```

For example to compile and run the code mpitest using SunHPC gnu (gcc based compiler) on 8 cores:

```
module load sunhpc/8.2.1/gnu
# 64 bit
mpif90 -O3 -o mpitest64 mpitest.f
ompisub 8 ./mpitest64
# 32 bit
mpif90 -m32 -O3 -o mpitest32 mpitest.f
ompisub 8 ./mpitest32
```

Unless you supply a 32 bit compiler switch, then the default is to compile 64 bit code. The following table shows the switches and modules available:

| Compiler | SunHPC Module | Compiler Module | 32 bit switch |
|---|---|---|---|
| Gnu | sunhpc/8.2.1/gnu | - | -m32 |
| Intel | sunhpc/8.2.1/intel | intel/compiler111 | -m32 |
| Pathscale | sunhpc/8.2.1/pathscale | - | -m32 |
| Pgi | sunhpc/8.2.1/pgi | pgi/9.0-4 | -tp=k8-32 |

The exact version numbers are correct at the time of writing, but may be newer on your system. Please check by executing the *module avail command. Codes compiled with any of the SunHPC packages can be submitted to SGE via the ompisub command.*

## 7.3 Mpich

*Streamline no longer support vanilla mpich, since this has been superceded by mpich2. Please see http://www-unix.mcs.anl.gov/mpi/. However some applications still require mpich built for Myricom's MX interconnect or the OpenIB Mvapich. This section applies mainly to these implementations of mpich. Before using mpich (myrinet mpich-mx, mpich-gm or mvapich over IB), make sure you have the correct environment. If you are using mpich then select the correct one for the interconnect and compiler you wish to use. For example:*

```
~$ module load  mpich/mx-1.2.6-INTEL
~$ which mpicc
/usr/local/mpich-mx-1.2.6-INTEL/bin/mpicc
~$ module load intel/compiler101_x86_64
~$ which ifort
/opt/intel/compiler101/x86_64/bin/ifort
```

### 7.3.1 compiling Mpich codes

*Use the correct mpich built in mpi wrapper. For example to compile the mpitest.f test code:*

```
~/benchmarks/mpi$ mpif90 -O3 -o mpitest mpitest.f
mpitest.f(24): (col. 12) remark: LOOP WAS VECTORIZED.
mpitest.f(96): (col. 9) remark: LOOP WAS VECTORIZED.
mpitest.f(52): (col. 9) remark: LOOP WAS VECTORIZED.
```

### 7.3.2 submitting Mpich jobs

*It is recommended you use mpichsub meta-script to submit mpich/mvapich jobs. For example:*

```
~/benchmarks> mpichsub 16 ./mpitest
```

## 7.4 Mpich2

*Please select the mpich2 environment to run mpich2 jobs. For example:*

```
~$ module load mpich2
~$ module load intel/compiler101_x86_64
~$ which mpif90
/usr/local/mpich2-GF90/bin/mpif90
```

*Before you start to run mpich2 jobs you need to create a .mpd.conf file under your home directory. This contains an arbitrary secret password (please DON'T use your login password) and must have the correct permissions:*

```
~$ cat .mpd.conf
secretword=MyBigSecret
~$ ls -al .mpd.conf
-rw------- 1 nick users 22 2008-03-12 10:24 .mpd.conf
```

*This allows the mpich2 mpd daemon ring to login to all the nodes used in a job.*

### 7.4.1 compiling Mpich2 codes

*By default mpich2 wrappers (mpif77, mpif90, mpicc, mpiCC) attempt to use the Gnu compiler suite. If you wish to use another compiler then you can add the -cc=, -CC= -fc= , -f90=, flags to select another C, C++, Fortran and Fortran 90 compiler as follows:*

```
mpicc  -cc=[C   compiler name]     [C   compiler options]
mpiCC  -CC=[C++ compiler name]     [C++ compiler options]
mpif77 -fc=[f77 compiler name]     [f77 compiler options]
mpif90 -f90=[f90 compiler name]    [f90 compiler options]
```

*For example to compile a fortran 90 code using mpich2 and the Intel compiler using the mpitest.f example program:*

```
mpif90 -f90=ifort -O3 -axT -o mpitest mpitest.f
```

### 7.4.2 submitting Mpich2 jobs

*It is recommended you use mpich2sub meta-script to submit mpich2 jobs.*

```
~/benchmarks> mpich2sub 16 ./mpitest
```

## 7.5 Mvapich 1 and 2

*Mvapich is a version of mpich supplied with the Open Fabrics Enterprise Edition (OFED) software for use with Infiniband. Please refer to the sections on Mpich and Mpich2 for general useage of these packages. In particular mpichsub and mpich2sub can be used to submit mvapich and mvapich2 jobs to the SGE queues. Both mvapich and mvapich2 come in 4 flavours according to their compiler support: Gnu, Intel, Pathscale and PGI. You cannot use the cc=/f90= syntax to select a compiler as with vanilla mpich2:*

| *Compiler* | *Mvapich module* | *Compiler Module* |
|---|---|---|
| *Gnu* | *mvapich/gcc/1.1.0* | - |
| *Intel* | *mvapich/intel/1.1.0* | *intel/compiler111* |
| *Pathscale* | *mvapich/pathscale/1.1.0* | - |
| *Pgi* | *mvapich/pgi/1.1.0* | *pgi/9.0-4* |

| *Compiler* | *Mvapich2 module* | *Compiler Module* |
|---|---|---|
| *Gnu* | *mvapich/gcc/2-1.2p1* | - |
| *Intel* | *mvapich/intel/2-1.2p1* | *intel/compiler111* |
| *Pathscale* | *mvapich/pathscale/2-1.2p1* | - |
| *Pgi* | *mvapich/pgi/2-1.2p1* | *pgi/9.0-4* |

# 8 HPC libraries: FFTW, Scalapack, Lapack, Blas, Atlas, MKL, ACML

## 8.1 FFTW

*The FFTW (Version 2) libraries (single precision, double precision, complex and real libraries are located in the standard library path : /usr/lib64/ ( static and dynamic libraries):*

```
/usr/lib64/libdfftw.a          /usr/lib64/libsfftw.a
/usr/lib64/libdfftw.so         /usr/lib64/libsfftw.so
/usr/lib64/libdrfftw.a         /usr/lib64/libsrfftw.a
/usr/lib64/libdrfftw.so        /usr/lib64/libsrfftw.so
```

## 8.2 Scalapack

*The dynamic and static Scalapack libraries are installed by default in /usr/local/lib64.*

## 8.3 Lapack/Blas

*Multiple versions of lapack and blas libraries are installed. This is because there are 4 main versions of the Blas library available for your system: Atlas, MKL, ACML, and Goto. Each Blas library has different license requirements and comes with a matching lapack library.*

## 8.4 Atlas

*A package is available from Streamline-Computing to compile up the Netlib Atlas Blas Lapack package. This runs as an SGE job on your system and prepares an optimal Blas/Lapack library tuned for your cluster. Atlas is freely available BSD-style licensed software. The Atlas build job may already have been run as part of system testing prior to shipping. In which case the Atlas/Lapack libraries are located in /usr/local/lib64/atlas . Please see http://math-atlas.sourceforge.net/ if you are unfamiliar with Atlas.*

## 8.5 MKL

*The Intel Math Kernel Library (MKL) is licensed software. If you have purchased a license for MKL as part of your cluster the 64 bit libraries will*

*be installed in /opt/intel/mkl/VERSION/em64t and the 32 bit libraries in /opt/intel/mkl/VERSION/32. ( Libraries for the Itanium architecture are in /opt/intel/mkl/VERSION/64.). Currently VERSION=10 .*

## 8.6 ACML

*ACML libraries are licensed from AMD. A free license can be obtained by registering at :*

`http://developer.amd.com/cpu/Libraries/acml/downloads/Pages/default.aspx .`

*By default the ACML libraries install into /opt/acml. A separate library is available for compatibility with each of the GNU, Intel, Pathscale and PGI compilers. These are found in the gfortran64, ifort64, pathscale64 and pgi64 sub-directories respectively.*

```
/opt/acml$ ls -d *64
gfortran64  ifort64  pathscale64  pgi64
```

*If you purchased a PGI compiler license you will also be able to use the acml library that comes with the PGI compiler suite and is located in the standard PGI library directory.*

## 8.7 Goto Blas

*The Goto Blas library is freely licensed to academic users. Non-academic users may obtain the library upon paying the license fee. To obtain a license and download the latest library for your architecture please see*

`http://www.tacc.utexas.edu/resources/software/#blas% .`

## 8.8 Linking code with Scalapack/Lapack/Blas

*This sub-section assumes you are running a modern cluster supporting gcc version 4 or above. The gcc 4 package contains the gfortran fortran 90 compiler which produces code with a single training underscore compatible with the Intel, PGI and Pathscale compilers. To check your gcc version use the gcc –version command. eg on SuSE SLES10_SP1 :*

```
~$ gcc --version | head -1
gcc (GCC) 4.1.2 20070115 (prerelease) (SUSE Linux)
```

*On RedHat EL4 and clones you can use the gcc4/gfortran non-native package. For the Pathscale compiler you may need to add the compiler option -fno-second-underscore. If you are linking a C code to the Fortran Scalapack/Lapack libraries it may be easier to use the appropriate fortran compiler to link the code since this will invoke the loader and link with any outstanding fortran libraries:*

```
# Linking a C code using a fortran loader
   gfortran          [link options] # GNU       compiler
   ifort -nofor-main [link options] # Intel     compiler
   pgf90 -Mnomain    [link options] # PGI       compiler
   pathf90           [link options] # Pathscale compiler
```

### 8.8.1 Atlas

*Please use the following link option (all compilers):*

```
  -L/usr/local/lib64/atlas -L/usr/local/lib64    \
  -lpthread -lm -lscalapack -llapack  -lmpiblacsCinit     \
  -lmpiblacs  -lcblas -lf77blas -latlas -lgfortran
```

### 8.8.2 MKL Version 11.1

*Please use the following link option (all compilers):*

```
   -L/opt/intel/Compiler/11.1/lib/intel64  \
   -L/opt/intel/Compiler/11.1/mkl/lib/em64t
   -L/usr/local/lib64  \
   -lscalapack -llapack -lmkl_intel_lp64 -lmkl_core \
   -liomp5 -lpthread -lgfortran
```

*Make sure the mkl/11.x/em64t environment module is loaded before running the code.*

### 8.8.3 ACML

*This is the link line needed using the built in acml library that comes with the PGI compiler suite (using PGI 7.1 in example):*

```
  -L/usr/local/lib64 -L/usr/pgi/linux86-64/7.1/libso
  -lscalapack -llapack -lacml -lpthread -lgfortran
```

### 8.8.4 Goto Blas

*This assumes you have installed the Opteron Goto Blas library in /usr/local/lib64 (libgoto_opt-64_1024-r0.97.so is an example, replace this with the actual Goto Blas library):*

```
 -L/usr/local/lib64 -lscalapack -lmpiblacsF77init -lmpiblacs \
 -lmpiblacsF77init /usr/local/lib64/libgoto_opt-64_1024-r0.97.so \
 -llapack  -lpthread -lgfortran
```

# 9 Case study: AMBER8 benchmark

*In order to show how all the previous sections in this guide come together, we provide an example of compiling and running a complex Chemistry code, AMBER8 which is a widely used application licensed from the the Scripps Institute:*

`http://amber.scripps.edu/`

*Streamline-Computing neither endorse this code, nor claim that the recipe provided here is optimal. We merely provide details of a working benchmark in order to illustrate the various steps required to get from a source code to a running parallel application on a Streamline cluster. This is based on support provided to a previous Streamline customer.*
*For the purposes of this illustration we will use the following setup:*

- *MPI : OpenMPI 1.2.6*
- *Compiler : Intel 10.1*
- *Libraries : Atlas Blas,Lapack*

## 9.1 Setting the environment

*The first step is to ensure the correct environment is loaded. In this example, this is done as follows:*

```
~$ module clear
~$ module load intel/compiler101_x86_64
~$ module load openmpi/1.2.6-1/intel
~$ module load atlas
```

## 9.2 Compiling the source code

*In the amber8 src directory there is a config.h script which controls the compile and link options. In our example the critical sections look like the following (config.h):*

```
#-------------------------------------------------------------------------
# Availability and method of delivery of math and optional libraries
#-------------------------------------------------------------------------
USE_BLASLIB=$(VENDOR_SUPPLIED)
USE_LAPACKLIB=$(VENDOR_SUPPLIED)
USE_LMODLIB=$(LMOD_UNAVAILABLE)
#-------------------------------------------------------------------
# C compiler
#-------------------------------------------------------------------
```

```
CC= mpicc
CPLUSPLUS=mpiCC
ALTCC=mpicc
CFLAGS=-O3 $(AMBERBUILDFLAGS)
ALTCFLAGS= -O3 $(AMBERBUILDFLAGS)
CPPFLAGS= -O3 $(AMBERBUILDFLAGS)

#-----------------------------------------------------------------------
# Fortran preprocessing and compiler.
# FPPFLAGS holds the main Fortran options, such as whether MPI is used.
#-----------------------------------------------------------------------
FPPFLAGS= -P -I$(AMBER_SRC)/include -DMPI  $(AMBERBUILDFLAGS)
FPP= cpp -traditional $(FPPFLAGS)
FC= mpif90
FFLAGS= -O3      $(LOCALFLAGS) $(AMBERBUILDFLAGS)
FOPTFLAGS= -O3   $(LOCALFLAGS) $(AMBERBUILDFLAGS)
FPP_PREFIX= _
FREEFORMAT_FLAG= -free
ATLAS=-L/usr/local/lib64/atlas -lcblas -lf77blas -latlas -llapack
#-----------------------------------------------------------------------
# Loader:
#-----------------------------------------------------------------------
LOAD= mpif90  $(LOCALFLAGS) $(AMBERBUILDFLAGS)
LOADCC= mpicc  $(LOCALFLAGS) $(AMBERBUILDFLAGS)
LOADLIB=  $(ATLAS)
LOADPTRAJ= mpif90  -nofor_main $(LOCALFLAGS) $(AMBERBUILDFLAGS)
```

*The parallel code is then compiled using the command*

```
make parallel
```

*In this example we are interested in the application called "sander" which is created in the Amber exe directory. It is convenient to move this and rename it :*

```
cd ../exe ; mv sander ~/bin/sander_openmpi_intel_atlas
```

*As a final check that the library paths are correct it is useful to use the ldd command on the new executable:*

```
~$ ldd ~/bin/sander_openmpi_intel_atlas | cut -f 1-3 -d " "
   libcblas.so => /usr/local/lib64/atlas/libcblas.so
   libf77blas.so => /usr/local/lib64/atlas/libf77blas.so
   libatlas.so => /usr/local/lib64/atlas/libatlas.so
   liblapack.so => /usr/local/lib64/atlas/liblapack.so
   libmpi_f90.so.0 => /opt/openmpi-1.2.6-1/intel///lib64/libmpi_f90.so.0
```

```
libmpi_f77.so.0 => /opt/openmpi-1.2.6-1/intel///lib64/libmpi_f77.so.0
libmpi.so.0 => /opt/openmpi-1.2.6-1/intel///lib64/libmpi.so.0
libopen-rte.so.0 => /opt/openmpi-1.2.6-1/intel///lib64/libopen-rte.so.0
libopen-pal.so.0 => /opt/openmpi-1.2.6-1/intel///lib64/libopen-pal.so.0
libdl.so.2 => /lib64/libdl.so.2
libnsl.so.1 => /lib64/libnsl.so.1
libutil.so.1 => /lib64/libutil.so.1
libm.so.6 => /lib64/libm.so.6
libpthread.so.0 => /lib64/libpthread.so.0
libc.so.6 => /lib64/libc.so.6
libgcc_s.so.1 => /lib64/libgcc_s.so.1
libgfortran.so.1 => /usr/lib64/libgfortran.so.1
libifport.so.5 => /opt/intel/compiler101/x86_64/lib/libifport.so.5
libifcore.so.5 => /opt/intel/compiler101/x86_64/lib/libifcore.so.5
libimf.so => /opt/intel/compiler101/x86_64/lib/libimf.so
libsvml.so => /opt/intel/compiler101/x86_64/lib/libsvml.so
libintlc.so.5 => /opt/intel/compiler101/x86_64/lib/libintlc.so.5
/lib64/ld-linux-x86-64.so.2
```

### 9.3 Running the code

*An amber test case called explct_wat has been used to provide benchmark tests for a range of processor counts 1,2,4,8,16,32, and 64. This test requires a number of input files and produces a number of output files. The sander code requires a number of arguments. In order to keep things clear we will run the test for each processor count in a separate directory. The following simple shell script, run_openmpi.sh, is used to provide the complete set of results:*

```
#!/bin/bash
# OpenMPI test
EXEC=$HOME/bin/sander_openmpi_intel_atlas
NAME=openmpi_intel_atlas
CPUS="64 32 16 8 4 2 1"
input="_2ps.infile"

ARGS="-i explct_wat.mmdin7 \
      -o explct_wat.mdout8 \
      -p explct_wat.prmtop  \
      -c explct_wat.restrt7 \
      -r explct_wat.restrt8 \
      -ref explct_wat.refc8 \
      -x explct_wat.mdcrd8 \
      -v explct_wat.vel8 \
```

```
      -e explct_wat.mden8 \
      -inf explct_wat.mdinfo"

for j in $CPUS ; do
   DIR=${j}_${NAME}
   rm -rf $DIR
   mkdir -p $DIR
   ( cd $DIR
     ln -s ../explct_wat* .
     ln -s explct_wat$input explct_wat.mmdin7
     ompisub $j $EXEC $ARGS
    )
done
```

*The code is using OpenMPI, so the ompisub meta-script is invoked in each directory to submit a job.*

```
~/benchmarks/AMBER8/TESTS/explct_wat> ./run_openmpi.sh
~/benchmarks/AMBER8/TESTS/explct_wat> qstat
job-ID  prior   name       user         state submit/start at     queue      slots ja-task-
-----------------------------------------------------------------------------------------
    708 0.60500 sander_ope sccomp       r     07/22/2008 10:15:42 parallel.q@comp03   8
    709 0.54786 sander_ope sccomp       qw    07/22/2008 10:15:34                     4
    710 0.51929 sander_ope sccomp       qw    07/22/2008 10:15:35                     2
    711 0.50500 sander_ope sccomp       qw    07/22/2008 10:15:36                     1
    712 0.50500 sander_ope sccomp       qw    07/22/2008 10:15:38                     1
    713 0.50500 sander_ope sccomp       qw    07/22/2008 10:15:39                     1
    714 0.50500 sander_ope sccomp       qw    07/22/2008 10:15:40                     1
```

*Finally we use the time from the job output scripts to create a summary report:*

```
 find . -name '*.sh.o*' -print -exec tail -2 {} \;
./64_openmpi_intel_atlas/sander_openmpi_intel_atlas.sh.o708
Time in seconds: 147 Seconds
============================================================
./32_openmpi_intel_atlas/sander_openmpi_intel_atlas.sh.o709
Time in seconds: 163 Seconds
============================================================
./16_openmpi_intel_atlas/sander_openmpi_intel_atlas.sh.o710
Time in seconds: 230 Seconds
============================================================
./8_openmpi_intel_atlas/sander_openmpi_intel_atlas.sh.o711
Time in seconds: 388 Seconds
============================================================
```

```
./4_openmpi_intel_atlas/sander_openmpi_intel_atlas.sh.o712
Time in seconds: 623 Seconds
=========================================================
./2_openmpi_intel_atlas/sander_openmpi_intel_atlas.sh.o713
Time in seconds: 1112 Seconds
=========================================================
./1_openmpi_intel_atlas/sander_openmpi_intel_atlas.sh.o714
Time in seconds: 2124 Seconds
=========================================================
```

## 10 Understanding SGE queues

*We strongly recommend using one of the Streamline meta-script for submitting parallel MPI jobs. If you do need to write parallel job scripts by hand you will need to understand how the SGE queues and parallel environments are set up. Streamline configures three basic queues on a standard cluster: serial.q, parallel.q and multiway.q. The parallel.q and multiway.q queues both support running of parallel jobs. The multiway.q is a special queue normally only used with certain commercial codes. Therefore we only discuss serial.q and parallel.q here.*

*The parallel.q supports many types of parallel application. For instance several different types of parallel MPI applications as well as shared memory (smp) jobs. Because different parallel applications start and stop the processes in different ways, the parallel.q supports several parallel environments (PE's). The user must select the correct PE when launching a parallel application. This is done using the qsub flag -pe [pename] followed by the slot count. For parallel.q the slot count is the number of compute nodes. Within a slot (node) a parallel application is allowed to run up to NCORES threads where NCORES is the number of cpus or cores.*

*The three basic queues are mutually exclusive in the following sense:*

*If any parallel job has processes running on any particular node, then no serial jobs are allowed on that same node. No two different parallel jobs are allowed to have processes running on the same node. The total number of serial jobs able to run on a single node is NCORES. If one or more serial jobs are running on a node then no parallel job is allowed to use the same node.*

## 11 Understanding SGE PE's

*An SGE job will run in the parallel.q if the job is submitted using the -pe pename option where pename is one of smp, mpich, mpich2, or openmpi, used for running shared memory (smp), Mpich MPI, Mpich2 mpi and Open-MPI jobs respectively. When an SGE job runs under any particular PE the*

*following actions take place:*

- *SGE produces a list of hosts $PE_HOSTFILE*
- *SGE executes a "start" script for the PE*
- *SGE runs the users job script*
- *On termination a "stop" script is executed*

*To locate the start and stop scripts, just list the appropriate SGE PE by using qconf -sp pename. Eg for the openmpi PE:*

```
~> qconf -sp openmpi
pe_name           openmpi
slots             256
user_lists        NONE
xuser_lists       NONE
start_proc_args   /usr/local/sge6.0/streamline/mpi/ompi_start.sh  $pe_hostfile \
                  $job_id
stop_proc_args    /usr/local/sge6.0/streamline/mpi/ompi_stop.sh $job_id
allocation_rule   $round_robin
control_slaves    TRUE
job_is_first_task FALSE
urgency_slots     min
```

*The $pe_hostfile is a list of nodes in SGE format which is available when the job runs. The $job_id is the Job ID of the job.*

*To examine the contents of the PE_HOSTFILE, you can use a simple script and submit it to the parallel.q, for example:*

```
~/benchmarks> cat ptest.sh
#!/bin/bash
#$ -V -cwd
echo "The Job ID of this job is $JOB_ID"
echo "The pe host file follows:"
cat $PE_HOSTFILE
```

*Notice that in the job script the variables PE_HOSTFILE and JOB_ID are in upper case. This is not a typing error.*

*Submit it using, for example, 4 slots:*

```
~/benchmarks> qsub -pe openmpi 4 ptest.sh
Your job 206 ("ptest.sh") has been submitted
~/benchmarks> cat ptest.sh.o206
The Job ID of this job is 206
The pe host file follows:
```

```
comp04 1 parallel.q@comp04 <NULL>
comp01 1 parallel.q@comp01 <NULL>
comp03 1 parallel.q@comp03 <NULL>
comp05 1 parallel.q@comp05 <NULL>
```

*For the default PE's setup by Streamline, the $PE_HOSTFILE is pre-processed to give a plain host list as follows:*

| *PE* | *Plain HOSTFILE* |
|---|---|
| *mpich* | *$HOME/.mpich/mpich_hosts.$JOB_ID* |
| *mpich2* | *$HOME/.mpich/mpich_hosts.$JOB_ID* |
| *openmpi* | *$HOME/.mpich/mpich_hosts.$JOB_ID* |
| *smp* | *NONE* |

*In order to write a manual parallel job script a user must therefore :*

- *Be aware of how to spawn a parallel job using a hostfile.*
- *Use the correct PE.*
- *Clean up job correctly at termination.*

*It can thus be appreciated that writing a parallel job script by hand is somewhat complicated and prone to error, which is why we recommend to use one of the Streamline meta-scripts described earlier for submitting parallel jobs.*

# 12 Further documentation

*This section lists the man pages and online support links for various packages.*

## 12.1 Compilers

### 12.1.1 Gnu

*[man,info] [gcc,g++,gfortran]*

### 12.1.2 Intel

*man [icc, icpc, ifort] .*
*Online support: http://softwarecommunity.intel.com/support/*

### 12.1.3 Pgi

*man [pgcc, pgCC, pgf77, pgf90, pgf95]*
*Online support: http://www.pgroup.com/support/index.htm*

### 12.2 SGE

*man [qsub,qdel,qstat,qhold,qrls,complex]*
*Pdf user manual: http://docs.sun.com/app/docs/doc/817-6117?a=load*
*( N1 Grid Engine 6 User's Guide ).*

### 12.3 OPenMP

*Various links and tutorial at:*
*http://openmp.org/wp/*

### 12.4 OpenMPI

*See online links at:*
*http://www.open-mpi.org/*

### 12.5 Mpich

*See online links at:*
*http://www-unix.mcs.anl.gov/mpi/mpich1/docs.html*
*For Myrinet mpich (mx,gm) see also:*
*http://www.myri.com/scs/*

### 12.6 Mpich2

*Online links at :*
*http://www.mcs.anl.gov/research/projects/mpich2*

### 12.7 Netlib

*The man pages for the Blas, Lapack and Scalapack fortran routines are in /usr/share//man/man3 on the front end server. For example man dggqrf describes the calling procedure for the lapack subroutine DGGQRF (Generalized QR Factorisation).*
*Online guides and FAQ's are available at:*
*http://www.netlib.org/blas/*
*http://www.netlib.org/lapack/*
*http://www.netlib.org/scalapack/*

### 12.8 FFTW

*See links at http://www.fftw.org/*

# 13 FAQS

# 14 Trouble shooting